



CSc 174

Database Management Systems

7. Database Programming Techniques

Ying Jin

Computer Science Department

California state University, Sacramento

How to interact with DB?

- How to create tables?
- How to insert, update, delete tuples?
- How to create views?
- How to specify queries?
- ◆ Interactive interface provided by DBMS.
- ◆ Convenient but not sufficient

Database Programming

- ◆ Objective: to access a database from an application program
- ◆ A majority of database operations are made through application programs
- ◆ e.g. web applications

Database Programming Approaches

◆ Embedded commands

- Database commands are embedded in a general-purpose programming language

◆ API

- Available to the host language for database calls; known as an *API*

Typical Sequence of Interaction in Database Programming

1. Client program **opens a connection** to the database server
2. Client program submits **queries** to and/or updates the database
3. When database access is no longer needed, client program **terminates** the connection

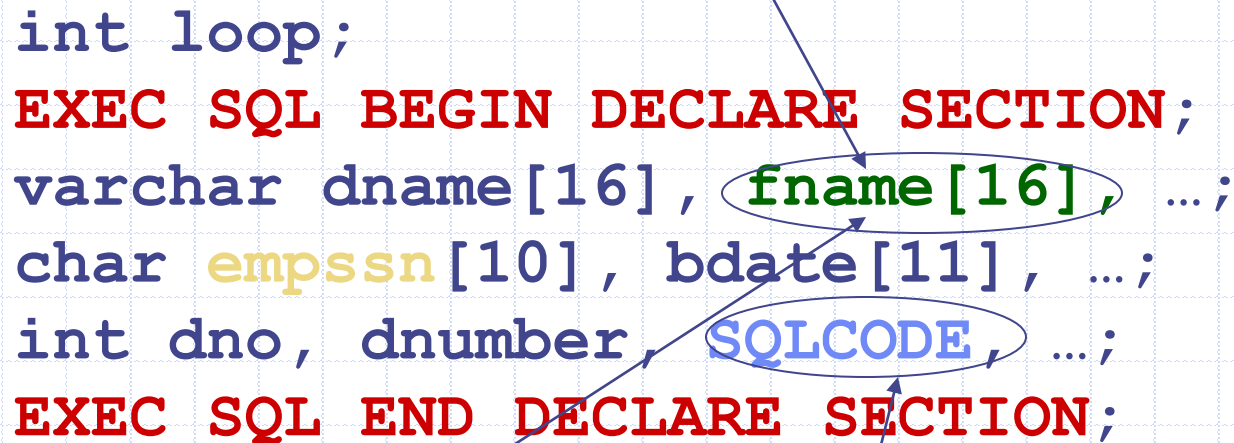
Embedded SQL

- ◆ Embedded SQL in a general-purpose *host* programming language
 - e.g. C, Java
- ◆ EXEC SQL, END-EXEC (or ;)
 - Distinguished from the host language
- ◆ :
 - *shared variables* (used in both languages)
 - prefix in SQL

Example: Variable Declaration in Language C

Variables inside DECLARE are shared

```
int loop;  
EXEC SQL BEGIN DECLARE SECTION;  
varchar dname[16], fname[16], ...;  
char empssn[10], bdate[11], ...;  
int dno, dnumber, SQLCODE, ...;  
EXEC SQL END DECLARE SECTION;
```



communicate errors/exceptions between the database and the program

Can appear in SQL (:fname) and C (fname)

Embedded SQL in C - programming Examples

- ◆ Read a ssn and then retrieves the employee tuple with that ssn from the database.

```
loop = 1;
while (loop) {
    prompt ("Enter SSN: ", empssn);
    EXEC SQL
        select FNAME, LNAME, ADDRESS, SALARY
        into :fname, :lname, :address, :salary
        from EMPLOYEE where SSN == :empssn;
    if (SQLCODE == 0) printf(fname, ...);
    else printf("SSN does not exist: ", empssn);
    prompt("More SSN? (1=yes, 0=no): ", loop);
}
```

SQLCODE==0: the previous statement was executed without errors.

Embedded SQL in C - programming Examples (Cont.)

◆ How many tuples are retrieved in

EXEC SQL

```
select FNAME, LNAME, ADDRESS, SALARY  
into :fname, :lname, :address, :salary  
from EMPLOYEE where SSN == :empssn;
```

?

Embedded SQL in C Using Cursor

- ◆ Process **multiple** tuples
- ◆ Declare a cursor
- ◆ OPEN CURSOR
 - Fetches the query result from the database
 - Sets the cursor to a position **before the first row** in the query result.
- ◆ FETCH
 - Move the cursor to the ***next*** tuple
- ◆ CLOSE CURSOR
 - Done with the processing the result of the query associate with that cursor
- ◆ SQLCODE > 0
 - Cursor past the last tuple

Embedded SQL - Example

//program segment E2:

0) prompt("Enter the Department Name:", dname);

1) EXEC SQL

2) Select DNUMBER into :dnumber

3) From DEPARTMENT where DNAME=:dname;

4) EXEC SQL DECLARE EMP CURSOR FOR

5) Select SSN, FNAME, MINIT, LNAME, SALARY

6) From EMPLOYEE where DNO= :dnumber

7) for UPDATE OF SALARY;

8) EXEC SQL OPEN EMP;

Embedded SQL – Example (Cont.)

```
9) EXEC SQL FETCH from EMP into :ssn, :fname, :minit, :lname,
    :salary;
10) While (SQLCODE ==0) {
11)   printf (" Employee name is:", fname, minit, lname);
12)   prompt ( "Enter the raise amount:", raise);
13)   EXEC SQL
14)       Update EMPLOYEE
15)       Set SALARY = SALARY + :raise
16)       Where CURRENT OF EMP; //current tuple referenced by
    the cursor is the one to be updated
17)   EXEC SQL FETCH from EMP INTO :ssn, :fname,:minit,
    :lname, :salary;
18) }
19) EXEC SQL CLOSE EMP;
```

Dynamic SQL

- ◆ Objective: executing new (not previously compiled) SQL statements at run-time
 - a program accepts SQL statements from the keyboard at run-time

Dynamic SQL: Example 1

```
EXEC SQL BEGIN DECLARE SECTION;  
varchar sqlupdatestring[256];  
EXEC SQL END DECLARE SECTION;
```

...

```
prompt ("Enter update command:", sqlupdatestring);  
EXEC SQL EXECUTE IMMEDIATE :sqlupdatestring;
```

Dynamic SQL: Example 2

EXEC SQL BEGIN DECLARE SECTION;

varchar sqlupdatestring[256];

EXEC SQL END DECLARE SECTION;

...

prompt ("Enter update command:", sqlupdatestring);

EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;

EXEC SQL EXECUTE sqlcommand;

If the command is to be executed multiple times in the program, it can be prepared only once.

SQLJ

- ◆ Embed SQL Commands in Java
- ◆ Historically, SQLJ was developed after JDBC
- ◆ A SQLJ translator converts SQL statement into Java
- ◆ Need to install a JDBC driver

SQLJ Example 1

-variable definition

- 1) string dname, ssn, fname, fn, lname, ln, badate, address;
- 2) char sex, minit, mi;
- 3) double salary, sal;
- 4) integer dno, dnumber;

SQLJ Example 1

-Program segment

```
1)ssn= readEntry("enter ssn:");
2) try{
3)     #sql{select fname, minit, lname, address, salary
4)         into :fname, :minit, :lname, :address, :salary
5)         from Employee where ssn=:ssn;}
6) }catch(SQLException se) {
7)     system.out.println("ssn does not exist:" +ssn);
8)     return;
9) }
10) System.out.println(fname+"" +minit + "" +lname +
    "" +address + "" +salary);
```

SQLJ Iterators

- ◆ SQL query returns more than one row in a Java Program
- ◆ Two types
 - Named Iterator: Data types and column names are specified
 - Positional Iterator: Only data types are specified

SQLJ Named Iterator Example

```
...  
#sql iterator ProjIter(String name, int num, String loc, int dn);  
...  
try{  
    ProjIter p=null;  
    #sql p={SELECT pname name, pnumber num, plocation loc, dnum dn  
              FROM project};  
    while (p.next()){  
        System.out.println(p.name()+p.num()+p.loc()+p.dn());  
    }  
    p.close();  
}  
...
```

in a position before 1st row
in the query result

Positional Iterator

```
#sql iterator ProjIter(String, int, String, int);
String name=null;
int num=0;
String loc=null;
in dn=0;
...
try{
  ProjIter p=null;
  #sql p={SELECT pname, pnumber, plocation, dnum FROM
    project};
  #sql {fetch :p into :name, :num, :loc, :dn};
  while (!p.endFetch()){
    System.out.println(name+num+loc+dn);
    #sql{fetch :p into :name, :num, :loc, :dn};}
  p.close();
}
...
```

Database Programming with Functional Calls (API)

- ◆ Two Function call interface
 - SQL/CLI (Call level Interface)
 - JDBC

SQL Call Level Interface (CLI)

- ◆ A part of the SQL standard
- ◆ Provides easy access to several databases within the same program
- ◆ Certain libraries (e.g., `sqlcli.h` for C) have to be installed and available

Java Database Connectivity

- ◆ JDBC: SQL connection function calls for Java programming
- ◆ A Java program with JDBC functions can access any relational DBMS that has a JDBC driver

Steps in JDBC Database Access

1. Set up environment
2. Import Java sql package `(import java.sql.*)`

3. Register JDBC driver

```
Class.forName  
("com.mysql.jdbc.Driver").getDeclaredConstructor().newInstance()
```

4. Connecting to the database

```
Connection conn =  
    DriverManager.getConnection(urlStr,username,password);
```

JDBC Database Access (Cont.)

5. Querying the Database

Three types of objects for querying DB:

- Statement
 - Execute SQL statements without any parameters.
- PreparedStatement
 - Execute one query multiple times
 - With input parameters.
- CallableStatement
 - Allows for SQL queries using stored procedures with input and output parameters.

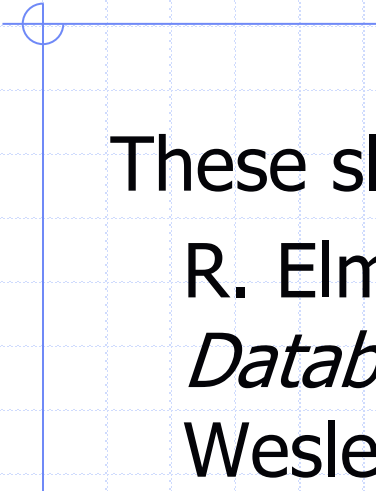
JDBC Database Access (Cont.)

-Examples

- Statement
- PreparedStatement
- CallableStatement

JDBC Database Access (Cont.)

6. Handler result sets
7. Close connection when finished retrieving information



These slides are based on following textbook :

R. Elmaseri and S. Navathe, *Fundamentals of Database System*, 7th Edition, Addison-Wesley.